Course Plan

- lec. 1: Introduction
- lec. 2: Mathematics for 3D Games
- lec. 3: Scene Graph
- lec. 4: Game 3D Physics +
- lec. 5: Game Particle Systems
- lec. 6: Game 3D Models
- lec. 7: Game Textures
- lec. 9: Game Materials
- lec. 8: Game 3D Animations
- lec. 10: Networking for 3D Games
- lec. 11: 3D Audio for 3D Games
- lec. 12: Rendering Techniques for 3D Games
- lec. 13: Artificial Intelligence for 3D Games

120

Changing the value of dt in Verlet

(whenever it's not constant)

Problem:

if dt now changes to a new dt'

then, all $\mathbf{p}_{old}$ must be updated to some $\mathbf{p}'_{old}$

Find $\mathbf{p}'_{old}$:

$$\vec{v} = (\mathbf{p}_{now} - \mathbf{p}_{old})/dt$$
$$\vec{v} = (\mathbf{p}_{now} - \mathbf{p}'_{old})/dt'$$

current velocity $\vec{v}$ and position $\mathbf{p}_{now}$ must not change

$$\Longrightarrow$$

$$\mathbf{p}'_{old} = \mathbf{p}_{now} \cdot (dt - dt')/dt + \mathbf{p}_{old} \cdot dt'/dt$$

121

Velocity damping in Verlet

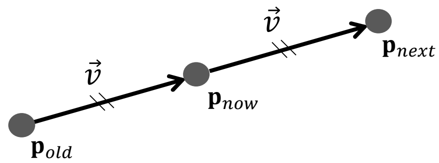
- Velocity at next frame: $\vec{v} = (\mathbf{p}_{next} - \mathbf{p}_{now})/dt$ (implicit)
- We want to multiply $\vec{v}$ a factor c_{DAMP} (e.g. 0.98 obtained as $(1\text{-}dt \cdot c_{\text{DRAG}})$)
  - before applying accelerations
- We can do that using a more general formula for $\mathbf{p}_{next}$

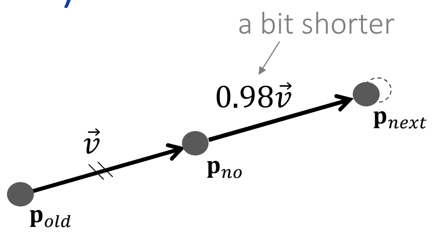
$$\mathbf{p}_{next} = 2 \cdot \mathbf{p}_{now} - 1 \cdot \mathbf{p}_{old} + dt^2 \cdot \vec{a}$$

$$\mathbf{p}_{next} = (1 + c_{\text{DAMP}}) \cdot \mathbf{p}_{now} - c_{\text{damp}} \cdot \mathbf{p}_{old} + dt^2 \cdot \vec{a}$$

123

Velocity damping in Verlet (geometric interpretation)





$$\mathbf{p}_{next} = 2 \cdot \mathbf{p}_{now} - 1 \cdot \mathbf{p}_{old}$$

Equivalently,
$\mathbf{p}_{next}$ is an extrapolation of $\mathbf{p}_{now}$, $\mathbf{p}_{old}$:

$$\mathbf{p}_{next} = mix(\mathbf{p}_{old}, \mathbf{p}_{now}, 2)$$

$$\mathbf{p}_{next} = 1.98 \cdot \mathbf{p}_{now} - 0.98 \cdot \mathbf{p}_{old}$$

Equivalently,
$\mathbf{p}_{next}$ is a different extrapolation of $\mathbf{p}_{now}$, $\mathbf{p}_{old}$:

$$\mathbf{p}_{next} = mix(\mathbf{p}_{old}, \mathbf{p}_{now}, 1.98)$$

124

Verlet integration + "Position Based Dynamics" (PBD)

init state

$\mathbf{p}_{now} \leftarrow \dots$

$\mathbf{p}_{old} \leftarrow \dots$

one step

$\vec{f} \leftarrow fun(\mathbf{p}_{now})$

$\vec{a} \leftarrow \vec{f}/m$

$\mathbf{p}_{next} \leftarrow 2\mathbf{p}_{now} - \mathbf{p}_{old} + \vec{a} \cdot dt^2$

Enforce constraints on $(\mathbf{p}_{next})$

$\mathbf{p}_{old} = \mathbf{p}_{now}$

$\mathbf{p}_{now} = \mathbf{p}_{next}$

125

Position Based Dynamics (PDB)

- A positional constraint is an equality/inequality involving the *positions* of particles.
  (a formula with '=' '>' '<' etc.)
  - Useful, for example, to model consistency conditions
  - Like "*solid objects don't compenetrate each other*", or "*steel bars won't become shorter or longer than they are*"
  - We will see many examples
- We enforce (impose) positional constraint directly by displacing the *positions* of particles
  - Thanks to Verlet: this displacement automatically causes some appropriate update of the velocity!
  - it's not necessarily correct, but it's plausible and robust

126

Verlet + Position Based Dynamics. Advantages

- flexibility: different constraints can be used to model many different phenomena
  - Useful constraints are straightforward to define
  - They are easy to impose (they involve only few particles)
  - They can be used to model many possible phenomena
  - See following slides for examples
- robustness : plausibility is ensured by *explicitly* enforcing the conditions we want to see
  - For example: a ball won't ever be seen outside the box containing it (at lest, not for many frames)
- No forces / impulses are needed to enforce any such consistency conditions
  - Which is what actually happens in the real world, but is more difficult to simulate robustly

127

Example of a positional constraint

«I want all particles to stay above ground (their y is not negative) »

Enforce this constraint: trivial!

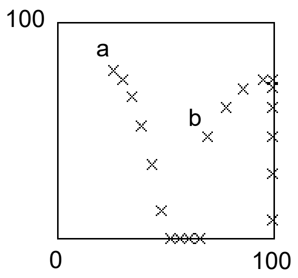
```
for (each particle i)
{
  if (p[i].y < 0) p[i].y = 0;
}
```

Imposing constraints like this one is a first part of collision response. For re-bounces, impulses must still be added (see collisions).

128

Example of a positional constraint (here, in 2D physics)

«I want particles to stay inside a 2D box [0 – 100] x [0 – 100] *»*



Enforce this constraint: simple clamp!

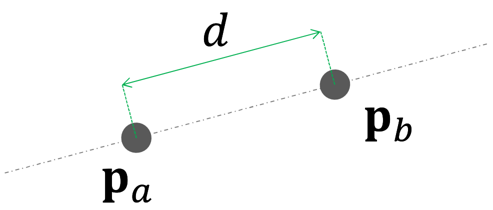
```
for (each particle i)
{
  p[i].x = clamp( p[i].x, 0, 100 );
  p[i].y = clamp( p[i].y, 0, 100 );
}
```

⚠ Imposing constraints like this one is a first part of collision response. For re-bounces, impulses must still be added (see collisions).

129

Example of positional constraint: equidistance constraint

*«Particles **a** and **b** must stay at a fixed distance **d** »*



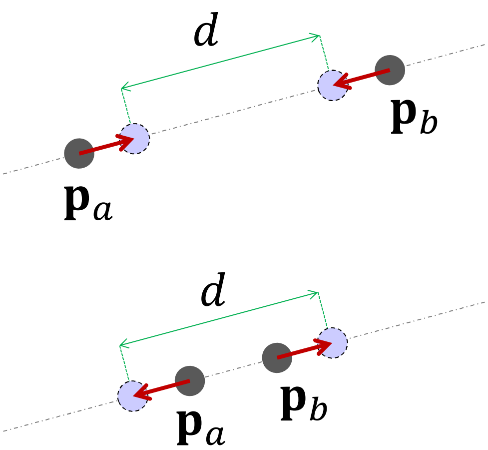
I want that... $\|\mathbf{p}_a - \mathbf{p}_b\| = d$

130

Enforce equidistance constraints (assuming equal masses)

if $\|\mathbf{p}_a - \mathbf{p}_b\| > d$

if $\|\mathbf{p}_a - \mathbf{p}_b\| < d$



131

Enforce equidistance constraints: pseudo code

```
Vector3 pa, pb; // curr positions of a,b
float d;        // distance (to enforce)

Vector3 v = pa - pb;
float currDist = v.length;

v /= currDist;  // normalization of v

float delta = currDist - d ;

pa += ( 0.5 * delta) * v;
pb -= ( 0.5 * delta) * v;
```

assuming equal mass, we move each particle *half the way*
(see later for the more general case)

132

Compare: equidistance constraints vs. springs

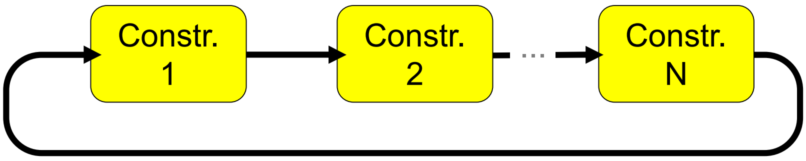
some constant scalar parameter l

- Similar
  - they both mean:
    these 2 particles "want to be" at *this* distance (not more, not less)
- but different
  - equidistance constraint:
    - applied during constraint enforcement
    - directly affects positions
    - models a rigid rod between the two particles
      - of a given length
    - sometimes called a "HARD" constraint
  - spring:
    - applied during force evaluation step
    - affects forces, therefore accelerations
    - models a deformable spring between the two particles
      - of a given length
    - sometimes called a "SOFT" constraint
- A physic engine can combine them in one object!

133

Enforcing sets of constraints

- There are many constraints to impose:
  when you solve one → maybe you break another!
- Simultaneous enforcement: computationally expensive
- Practical & easy solution: just enforce them in cascade
  (similar in concept to Gauss-Seidel solvers):



Repeat until convergence (= max error below threshold)
...but at most for *N* times! (reminder: our simulation is *soft* real-time)

134

Enforcing sets of constraints one after the other (in cascade)

- The whole loop for imposing the constraints happen in the constraint enforcement phase on one physics step!
- Notes about convergence:
  - needed iterations (typically) few: e.g. 1 ~ 10 (efficient!).
  - if convergence not reached within a given number of steps: never mind, next frames will fix it (it’s fairly robust)
  - (it is never reached, if constraints are contradictory)
  - Optimization (to reduce the number of needed iterations): solve the most unsatisfied constraints first

⚠ Problem: it’s a sequential approach! ☹
  - parallelized versions (similar to Jacobi solvers) are possible
  - they have a worse convergence in practice (they require more iterations), but each iteration is faster

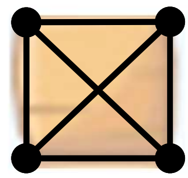
135

Compounds of particles disguised as rigid bodies

140

Combining equidistance constraints we obtain rigid objects

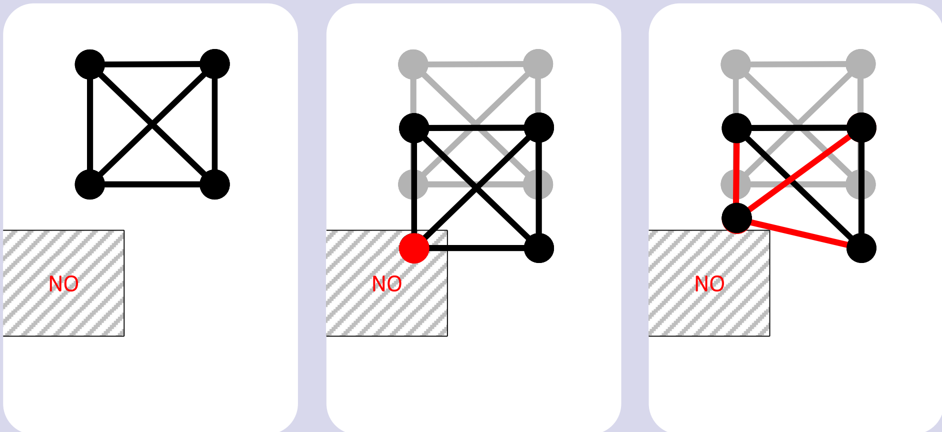
- Rigid body dynamics as emerging behavior
  - without explicitly keeping track their orientation, angular vel, angular acc., etc.



A box?
(rigid object)
In 2D a configuration of:
- 4 particles
- 6 equidistance constraints

141

Example



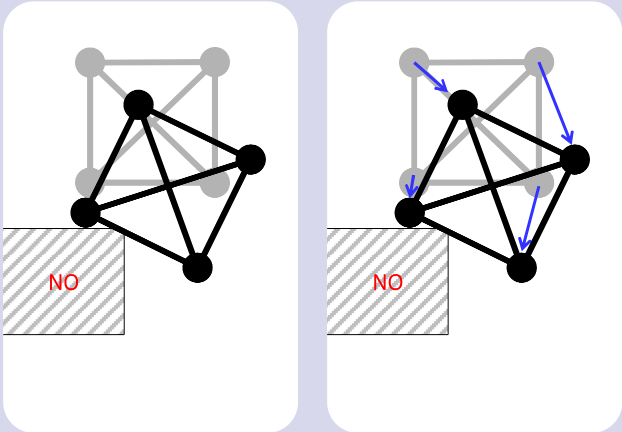
FRAME 0

FRAME 1
before constraints

FRAME 1
after 1st constraint

142

Example



FRAME 1
after all constraints
multiple times

FRAME 1
resulting
(implicit) velocities

In total: the "box",
under gravity + collision
- had rotated
- gained angular velocity (will keep rotating by inertia)

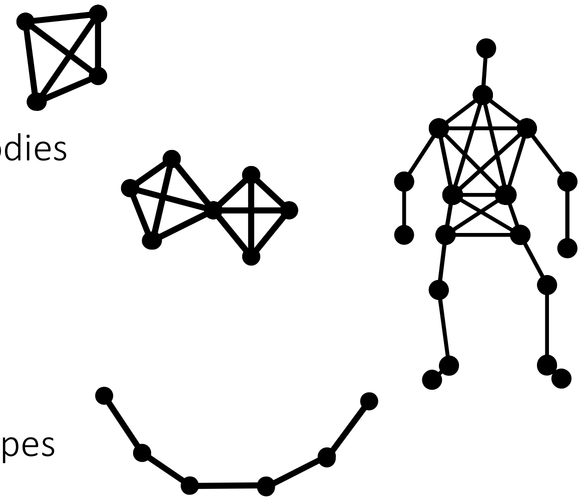
even the system does not (explicitly) handle rotations or angular velocities

(works in 3D as well!)

143

We can combine equidistance constraints to obtain…

- Rigid bodies
- Articulated bodies
- Ragdolls
- Cloth
- Non-elastic ropes
- …and more



145

Enforcing a positional constraint:
(assuming for now all particles have same mass)

- Test: does the equality/inequality hold?
- If so: nothing to do!
- Else:
  - All particles must be displaced a bit, so that it will
  - Infinite ways to achieve this. Which one to pick?
  - Answer:
    minimize the sum of *squared* displacements
    (with respect to current position)
  - Find the minimizer by analytically
    solving simple math problems
    ("analytically" = in closed form = "with formulas")

146

Enforcing positional constraints
(assuming for now all particles have same mass)

- We want to enforce a constraint $\mathcal{C}$ on particles $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$ in positions $\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}$ …
- $\mathcal{C}$ defined as an equality/inequality of $\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}, \ldots$:

$$\mathcal{C}: (\mathbf{p}_\mathrm{a}, \mathbf{p}_\mathrm{b}, \mathbf{p}_\mathrm{c}, \ldots) \rightarrow \{\, true \,, false \,\}$$

- We must apply the displacements $\vec{d}_\mathrm{a}$, $\vec{d}_\mathrm{b}$, $\vec{d}_\mathrm{c}$ found by:

$$\underset{\vec{d}_\mathrm{a}, \vec{d}_\mathrm{b}, \vec{d}_\mathrm{c}, \ldots}{\operatorname{argmin}} \left(\|\vec{d}_\mathrm{a}\|^2 + \|\vec{d}_\mathrm{b}\|^2 + \|\vec{d}_\mathrm{c}\|^2 + \cdots \right)$$

$$\text{such that } \mathcal{C}(\mathbf{p}_\mathrm{a} + \vec{d}_\mathrm{a}, \mathbf{p}_\mathrm{b} + \vec{d}_\mathrm{b}, \mathbf{p}_\mathrm{c} + \vec{d}_\mathrm{c}, \ldots)$$

among all the choices that satisfy this,
we want the one which minimizes this

147

Positional constraint example: "please don't sink under a general plane"

- We want to enforce the constraint
  "particle a must be above a given constant plane"
  - Given: position of the particle $\mathbf{p}_a$ and its mass m_a
  - A plane given by a point on it $\mathbf{p}_q$ and its normal $\hat{n}_q$
- We need to apply the displacement $\overrightarrow{d_a}$ found by minimizing:

$$\underset{\overrightarrow{d_a}, \overrightarrow{d_b}}{\operatorname{argmin}} \left(\left\| \overrightarrow{d_a} \right\|^2 \right)$$

$$\text{such that } \left\| (\mathbf{p}_a - \mathbf{p}_q) \cdot \hat{n}_q \right\| > 0$$

- And the solution (in closed form) is, trivially…

148

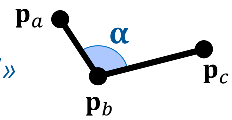
In pseudocode

```
Vector3 pa; // curr positions of a
float ma;   // mass (no effect here)
Vector3 pq; // point on the plane
Vector3 nq; // normal of the plane (unit)

Vector3 v = pa – pq;
float currDist = Vector3.dot( v , n );

if (currDist < 0.0)
   pa -= currDist * n; // just project!
else {} // constrain ok, nothing to do
```

149

More examples of possible positional constraints

- Preserve volume of some object: *«Volume is* v_{CONST} *»*
  - How to impose it:
    1. Estimate current total volume v
    2. uniform scale the entire object by factor $\sqrt[3]{v_{\text{CONST}} / v}$
- Fixed positions: *«particle* a *stays in* $\mathbf{p}_\mathbf{a}$ *»*
  - the particle is "pinned" in position
  - trivial to impose, but still useful!
- Angle constraints, e.g. $\boldsymbol{\alpha} < \boldsymbol{\alpha}_{\text{max}}$
  - e.g., on joints: *«elbows cannot bend backward»*



- Coplanarity / collinearity
  - «these N particles must stay on a line / on a plane»
- Non interpenetration
  - part of collision handling – see collisions later

150

Position Based Dynamics (PBD) summary

- A general approach for computing dynamics
- Ingredients:
  1. Use Verlet integration on particles
     - velocities are implicit
     - changes in positions induce changes in velocities
  2. Implement positional constraints on particles
     (e.g., equidistance constraint) to model things like:
     - Rigid bodies
     - Articulated / non rigid bodies
     - Non penetration (maybe, add collision impulses, see later)

151